Name: _____

Computer Science II — Fall 2021

Exam #1

This exam should have five pages. Closed book and notes.
No computers or calculators allowed.

Problem 1: [12 points]

What's the difference between an abstract class and an interface? Give an example of when you'd use each.

Problem 2: [12 points]

Thanks to subtyping, a *variable* of type `BasicDie` can instead refer to an *object* of type `HistoryDie` or any other subclass of `BasicDie`. In general, how does the Java compiler know it's safe to substitute an instance of a subclass for an instance of the original class?

Problem 3:

The questions below have to do with the following method, which could be added to the code you worked with on the `DieInheritance` project in lab:

```
public static int rollDice(ArrayList<BasicDie> dice) {
    Random rng = new Random();
    int total = 0;
    while(dice.size() > 0) {
        int index = rng.nextInt(dice.size());
        BasicDie d = dice.get(index);
        total = total + d.roll();
        dice.remove(index);
    }
    return total;
}
```

a) **[8 points]** When considering the number of computational steps required to run the method, what determines the problem size? In other words, what's *n* for this method?

b) **[15 points]** Come up with a reasonable estimate of *T(n)* for `rollDice` that represents the worst-case number of computational steps required to run the method. For full credit, list the steps that you're counting and be clear about whether they're happening once, or once per iteration of the loop. (You can underline them above if you prefer — single underline for once, double underline for operations that happen once per iteration.)

c) **[15 points]** What's *O(n)* for `rollDice`? Justify your answer by providing appropriate values for *c* and n_0 given your *T(n)* from above.

d) **[16 points]** Rewrite the method above so that it catches any exceptions that `roll()` might throw, but does so in such a way that `rollDice` still runs to completion and returns a total even if one or more dice in the input list consistently produce exceptions when rolled.

Problem 4: [22 points]

The `Controller` class from your last assignment was fancier than the `Remote` we used with the `Radio`, `TV`, and `Thermostat` classes. `Controller` had buttons for left and right as well as up and down, for example. We could make a version of `Radio` that worked with the fancier `Controller` if we wanted to though. (Then it would work with either the original `Remote` or the fancier `Controller`!)

Below, define a subclass of `Radio` called `BetterRadio`. It should implement the `Controllable` interface, but where the `left()` method accomplishes the same thing as `down()` in the original class, `right()` accomplishes the same thing as `up()`, and the `function()` method does the same thing as `reset()`. For full credit, write as little code as possible to achieve these goals. The original `Radio` code is on a page at the end of this exam, along with the `Controllable` interface.

```
/**
 * The Radio class models a simple FM radio tuner.  It remembers
 * which frequency it's tuned to, and has methods for moving up
 * or down through frequencies. (Assume the original interface it
 * implemented was called Remoteable.)
 */
public class Radio implements Remoteable {
    protected int freq;
    public static final int MAX_FREQ = 1079;    // 107.9 MHz
    public static final int MIN_FREQ = 875;     // 87.5 MHz

    public Radio() {
        freq = MIN_FREQ;
    }

    public Radio(int startFreq) {
        freq = startFreq;
    }

    public void up() {
        freq = freq + 2;
        if (freq > MAX_FREQ) {
            freq = MIN_FREQ;
        }
    }

    public void down() {
        freq = freq - 2;
        if (freq < MIN_FREQ) {
            freq = MAX_FREQ;
        }
    }

    public void reset() {
        freq = MIN_FREQ;
    }

    public String toString() {
        return ""+freq/10.0;
    }
}


/**
 * The Controllable interface lists the five methods that a class
 * must implement to be considered "Controllable".
 */
public interface Controllable {
    public void up();
    public void left();
    public void right();
    public void down();
    public void function();
}
```